Express 框架

简介

1. Express 是基于 Node.js 的极简 web 框架，它的运用十分的广泛。
2. Express 是一个路由和中间件 web 框架，其自身只有最低程度的功能：Express 应用程序基本上是一系列中间件函数调用。
3. npm 安装：npm install express --save

在 Express 中提供静态文件

1. 为了提供如图像、css 文件和 JavaScript 文件之类的静态文件，请使用 Express 中的 express.static 内置中间件函数。
2. 将包含静态资源的目录的名称（即静态资源的根路径）传递给 express.static 中间件函数，以便开始直接提供这些文件。例如，使用以下代码在名为 public 的目录中提供图像、css 文件和 JavaScript 文件。
3. 使用 app.use(express.static('rootpath'))加载静态资源
4. use() 方法的第一个参数可以指定一个虚拟路径。
5. 可以指定多个目录作为静态资源目录，只需要将 use()重复调用多次即可。

Express 路由

基础路由

1. 路由：根据请求路径和请求方式进行路径分发处理
2. http 常用的请求方式：
   - post 添加操作
   - get 查询操作
   - put 更新操作
   - delete 删除操作
   - get 为 url 地址栏传递参数，而 post、 put、和 delete 则通过请求体（body）传递参数。
3. restful api ： 一种特定格式的 URL
4. app.route(): 为路由路径创建可连接的路由处理程序。因为在单一位置指定路径，所以可以减少冗余和输入错误。参考下面的代码：

5. express.Router: 使用 express.Router 类来创建可安装的模块化路由处理程序。Router 实例是完整的中间件和路由系统；因此，常常称其为“微型应用程序”。

express 整合模板引擎

1. 在项目中安装对应的模板引擎 npm 包：
2. 设置以下运用程序设置：
    - views: 模板文件所在的目录。例如：app.set('views', './views')。注意第一个参数是固定的写法 'views',第二个参数是模板文件所在的根路径
    - view engine: 要使用的模板引擎。例如：app.set('view engine','pug')。 第一个参数是固定的写法，第二个参数是模板文件的后缀名。
3. 使 express 兼容模板引擎 例如：app.engine('art',require('express-art-template'));
4. 通过 res.render('模板名称',data);来渲染模板。

express 中间件

1. 中间件就是处理过程的一个环节，其本质是一个函数。
2. 中间件函数能够访问**请求对象**（req),**响应对象**（res)以及应用程序的请求/响应循环中的下一个中间件函数。下一个中间函数通常由名为 next 的变量表示。
3. 中间件函数可以执行以下任务：
    - 执行任何代码
    - 对请求和响应对象进行更改
    - 结束请求/响应循环
    - 调用堆栈中的下一个中间件函数
4. 如果当前中间件函数没有结束请求/响应循环，那么它必须调用 next()，以将控制权传递给下一个中间件函数。否则，请求将保持挂起状态。
5. Express 应用程序可以使用以下类型的中间件
    - 应用层中间件
    - 路由层中间件
    - 错误处理中间件
    - 内置中间件
    - 第三方中间件

应用层中间件

1. 使用 app.use()和 app.METHOD()函数将应用层中间件绑定到应用程序对象的实例，其中 METHOD 是中间件函数处理的请求方法的小写（如 get，post 等）

路由器层中间件

1. 路由器层中间件的工作方式与应用层中间件基本相同，差异之处在于它绑定到 express.Router()的实例
2. 使用方式：const router = express.Router();，使用 router.use()和 router.METHOD() 函数装入路由器层中间件。

错误处理中间件

1. 错误处理中间件函数的定义方式与其他中间件函数基本相同，差别在于错误处理函数有四个变量(err,req,res,next)；必须提供四个变量，即使无需使用 next 对象，也必须指定，否则 next 对象将被解析为常规中间件，从而无法处理错误。
2. 请在其他 app.use() 和路由调用之后，最后定义错误处理中间件。

内置中间件模块

1. 自 v.x 起，Express 不再依赖于 Connect。除了 express.static 外，Express 随附的所有中间件函数以单独模块的形式提供。

第三方中间件

1. 使用第三方中间件向 Express 程序添加功能需要先安装该中间件
2. 安装方式如：npm install cookie-parser --save

Express 参数处理

1. 使用 body-parser 第三方中间件，安装方式：npm install body-parser
2. get 请求使用 req.query 获得提交的数据
3. post 请求使用 req.body 获得提交的数据
4. Express 处理 restful api 形式 url 的参数：
    - 使用 req.params 属性得到参数对象，然后根据需要取出想要的数据即可

Express 后台接口开发

json 接口开发

1. Express 框架开发 json 接口十分简单，使用响应方法：res.json(); 将数据传递给客户端即可。
2. 参考下面的代码：

```
 //========== Express 框架开发后台接口 ==============
const express = require('express');
const app = express();
const database = require('./database');

//指定 api 路径 （json 接口）
app.get('/all-book',function (req,res) {
    database.getAllData('book', function (result) {
            res.json(result);
    });
});

app.listen(3002, function (err) {
    console.log('server running');
});
```

jsonp 接口开发

1. Express 框架的 jsonp 接口开始与 json 接口开发十分的类似，使用 res.jsonp(); 方法即可。
2. 默认情况下 Express jsonp 的回调函数的名称为 callback，可以通过 app.set('jsonp callback name',name); 来改变 jsonp 回调函数的名称。

参考下面的代码：

```
//========== Express 框架开发后台接口 ==============
const express = require('express');
const app = express();
const database = require('./database');

// 指定 api 路径 （jsonp 接口）
```

```
app.set('jsonp callback name', 'jsonp');
app.get('/all-book', function (req, res) {
    database.getAllData('book', function (result) {
            res.jsonp(result);
    });
});

app.listen(3002, function (err) {
    console.log('server running');
});
```

Express 框架解决跨域问题

1. 当我们在浏览器的 console 控制台中看到类似如下的信息：
2. Access to XMLHttpRequest at 'http://localhost:3001/books' from origin 'http://localhost:63342'
3. has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.

   这表示存在着跨域问题。解决的办法就是在拦截每个请求的路由上添加 'Access-Control-Allow-Origin' header。

4. 下面是 node express 框架解决跨域问题的代码，注意解决跨域的代码要放在处理路由的代之前，否则将会不起作用：

```
//===== 项目入口文件 =====
const express = require('express');
const app = express();
const bodyParser = require('body-parser');
const router = require('./router');

//----- 配置跨域 -----
// express 框架解决跨域问题的代码，注意该代码要放在 app.use(router); 之前
app.all('*', (req, res, next) => {
    res.header("Access-Control-Allow-Origin", "*");
    res.header("Access-Control-Allow-Headers", "X-Requested-With");
    res.header("Access-Control-Allow-Methods",
"PUT,POST,GET,DELETE,OPTIONS");
    res.header("X-Powered-By", 'Express');
    res.header("Content-Type", "application/json;charset=utf-8");
```

```
    next();
});

//----- 配置路由 -----
app.use(router);

//----- 配置参数解析-----
app.use(bodyParser.urlencoded({extended:true}));
app.use(bodyParser.json());

//----- 监听端口 -----
const port = process.env.port || 3001;
app.listen(port,(error) => {
    console.log(`server running on ${port}`);
});
```